**Corso di Programmazione Web e Mobile**

**A.A. 2017-2018**

**Planet Defender Game**

Giovanni Sorgente [911248]

# Planet Defender

Una game app basata sul concetto di RTS (Real Time Strategy)

# Introduzione

Planet Defender, vuole essere una game app, basata su Angular 6 che si prefigge l’obiettivo di infoltire il parco applicazioni ludico fruibile sia da browser che da smartphone (Android).

La direttiva che sta alla base del progetto è quella di creare una sessione di gioco in modalità arcade, che permetta all’utente, in maniera del tutto gratuita di distrarsi e riempire quei “tempi morti” che spesso capitano durante la giornata.

Ad oggi, troviamo nei vari store sempre più applicazioni che presentano contenuti solitamente ridotti ma dall’obiettivo mirato al “divertimento” e svago temporaneo dell’utente.

Prendiamo ad esempio i casi più eclatanti che possono tornare alla mente, come Candy Crush oppure il meno famoso Flappy Bird; la richiesta è quella di avere un accesso veloce alla modalità di gioco, eventuali perks dovuti alla partita interrompibile in qualsiasi momento ed uno stretto ciclo di vita della partita che può essere ripetuta infinitamente rilassando l’utente che in quel frangente non dovrà impegnare troppo intensamente le proprie capacità intellettive ma potrà comunque trovare soddisfazione dall’attività svolta risolvendo i piccoli e semplici enigmi posti in essere dal gioco stesso.

A mio avviso la vastità del parco software presente online oggi permette sicuramente di usufruire di qualsiasi tipologia di applicazione si stia cercando; ovvero credo sia molto complicato trovare idee innovative che siano davvero originali e che possano essere poi proposte al grande pubblico.

Sono piuttosto convinto che la continua ricerca da parte dell’essere umano di qualcosa di diverso, di nuovo, di mai visto; porti poi, anche in questo ambito, al medesimo risultato del non rendere mai del tutto coperta una determinata tipologia di applicazioni.

Ecco che una piccola variante può così dal nulla creare una nuova versione che stimola di nuova la voglia di spendere del tempo da parte dell’utente nel superare quel determinato livello o sconfiggere quel determinato avversario.

Di seguito una breve descrizione del progetto che ho chiamato: **Planet Defender**.

* + Si tratta di un gioco (fruibile sia da mobile che da web browser)
  + E' previsto un login e relativa registrazione
  + L'utente (che visto l'ambito applicativo per comodità definisco giocatore), una volta effettuato il login, può decidere di cercare una partita in attesa oppure ospitarne una nuova ed attendere l'ingresso di un nuovo partecipante
  + L'iter di gioco si svolge in questo modo:
    - Esiste un'area virtuale (mappa) dove si sfidano due giocatori per volta, un "difensore" (giocatore in "casa") ed un giocatore "attaccante" (giocatore "fuori casa")
    - La mappa è definita da N tasselli in larghezza ed N in altezza
    - Ogni giocatore ha X carri armati a disposizione
    - Ogni carro armato ha un determinato numero di bonus vita
    - Prima di iniziare il turno:
      * ognuno dei giocatori posiziona le sue truppe sulla mappa. (premerà un pulsante per indicare di essere pronto per la partita)
      * inoltre il giocatore di casa posiziona N edifici da difendere durante la partita
    - Non vi saranno turni specifici per i giocatori, la fase di gioco si svolgerà infatti in tempo reale
    - Il giocatore può decidere di:
      * muovere il carro armato per un massimo (valore da definire) numero di tasselli
      * attaccare un carro armato avversario
      * se giocatore "attaccante" può decidere di attaccare anche gli edifici
    - Vincerà:
      * il giocatore "difensore" se riuscirà per primo ad eliminare la minaccia di attacco eliminando tutti i carri armati dell'"attaccante"
      * il giocatore "attaccante" se riuscirà per primo ad eliminare tutti gli edifici posizionati dal "difensore"

## Breve analisi dei requisiti

### Destinatari

Possiamo considerare l’applicazione come rivolta a chiunque. Questo per la bassissima curva necessaria per l’apprendimento delle funzionalità base proposte dall’interfaccia.

Ad ogni modo è probabile che il target si riferisca poi a persone di giovane età che solitamente si approcciano a questa tipologia di applicazioni.

**Linguaggio.**

Portando inoltre una grafica composta pressoché da icone ed immagini, è sicuramente necessario studiare nel dettaglio quanto ogni immagine scelta sia in grado di comunicare alla persona che utilizza l’applicazione; possiamo però essere certi del fatto che avendo poche azioni ed uno scopo univoco semplice e diretto che sarà molto semplice per l’utente perseguire l’obiettivo della partita.

**Motivazione.**

Come già anticipato, si tratta principalmente di un’applicazione volta unicamente ad intrattenere l’utilizzatore,

AAAAAAA

### Modello di valore

Non è semplice stimare il valore dell’applicazione, una volta convertita in app mobile e portata sullo store, probabilmente con l’aggiunta di pubblicità potrebbe portare dei ricavi utili.

L’applicazione non da comunque un vero e proprio servizio, se non quello di poter intrattenere e rilassare il giocatore.

L’applicazione rimane molto pulita e dall’idea semplice e lineare. L’unica leva possibile a meno di aggiungere ulteriori funzionalità future, sarà quella di ottenere delle buone statistiche e cercare di acquisire la testa della classifica acquisendo il maggior numero di vittorie possibili.

L’intrattenimento è certamente un altro valore aggiunto che come anticipato poco prima potrebbe essere la chiave di volta per creare una nuova applicazione, simile alle precedenti, ma che possa comunque offrire qualcosa di nuovo o di mai provato e che porti degli utenti sulla piattaforma per un periodo medio breve durante il quale si potrà cercare di massimizzare i ricavi attraverso gli introiti delle pubblicità.

I tempi di sviluppo sono certamente bassi e c’è inoltre da considerare, che valutando il modello di sviluppo adottato si potrà produrre con uno sforzo minimo un’applicazione similare in tempi ancora piu brevi.

Killer APP

### Flusso dei dati

Quale qualità, stile e livello di dettagli caratterizzano i contenuti?

**Ottenere i contenuti**.

I contenuti vanno prodotti o possono essere reperiti sul mercato?

Quali sono i costi di produzione o riadattamento?

**Archiviare e organizzare i contenuti**

Quale metodo di archiviazione?

Quali metodi di selezione dei contenuti?

**Pubblicare i contenuti**

Quali trasformazioni devono essere applicate?

I contenuti devono essere resi in formati che facilitino la condivisione (riuso) o questa deve essere sfavorita?

Con che frequenza devo aggiornare i contenuti?

Come scelgo un contenuto?

Ci sono vincoli alla pubblicazione (diritti, licenze, scadenza)?

### Aspetti tecnologici

Pensando all’implementazione dell’idea sopra descritta, probabilmente l’unica strada ad oggi percorribile, per avere un time to market accettabile è quella di utilizzare delle tecnologie multi piattaforma che permettano la flessibilità di distribuzione ed utilizzo da parte dell’utente.

Si pensi ad esempio alla vastità di versioni browser (desktop/mobile) ed alle versioni smartphone esistenti. Senza delle librerie e frameworks adeguati il tempo di sviluppo sarebbe così lungo che porterebbe a dei costi iniziali troppo alti per lanciare sul mercato una nuova idea applicativa.

Detto questo, si è deciso di propendere per tutte quelle tecnologie innovative che ad oggi la fanno da padrone in merito allo sviluppo web e mobile.

Di seguito vado ad elencare tutte le principali tecnologie scelte, tentando, per ognuna di darne una breve descrizione ed identificarne poi i principali punti di forza.

**Json**

Javascript Object Notation, credo che sia stato in grado in pochi anni di scalzare senza alcun dubbio Xml per gli scopi di serializzazione di dati.

Possiamo dare alcune brevi caratteristiche:

* Schemaless
* Poco verboso
* In grado di rappresentare qualsiasi struttra ad albero

Standard definito nel .. da Douglas Crockford che ancora oggi mantiene su GitHub un repository per un parser JSON scritto in javascript.

Non avendo schema, possiamo dire che è debolmente tipizzato, quindi la differenza in genere tra i tipi è data dal contenuto stesso della foglia (property dell’oggetto).

**Bootstrap**

Ad oggi è semplicemente lo standard de facto per la realizzazione degli scheletri di pagine web e mobile.

Si tratta di una suite di librerie css e javascript che permette di generare in maniera molto semplice e rapida un layout responsive senza preoccuparsi delle risoluzioni dell’ambiente ospitante.

Per layout responsive intendiamo una grafica che sia in grado di adattarsi dinamicamente alle principali risoluzioni dei dispositivi ad oggi presenti sul mercato.

L’avvento di una libreria di questo tipo accelera considerevolmente i tempi di sviluppo permettendo, cosa di non poco conto anche a sviluppatori puri, senza un’anima dedita al disegno di interfacce di essere comunque efficaci e validi nel presentare i propri contenuti.

Javascript

Linguaggio di scripting creato da Brendan Eich nel 1995; a mio parere uno dei più flessibili e comodi linguaggi presenti ad oggi per la creazione di applicazioni di vario genere.

Pur essendo nato principalmente ai tempi con l’obiettivo di costruire script dinamici nei primi browser nascenti, oggi si afferma sempre più come linguaggio utilizzato anche per lo sviluppo di servizi con il noto NodeJs.

Typescript

Definito in genere come un “super set” di istruzioni costruito sopra Javascript,

Snippets e modalità di utilizzo

<creazione api, modello codice backend, … >

***Riferimenti***

Angular 6/Angular-cli

Angular ad oggi in versione 6, nasce nel 2010 con una prima versione scritta interamente in javascript che si basava fortemente sul pattern MVC.

L’idea alla base del framework è quella di poter definire modelli dati, viste e controllori ben disaccoppiati tra loro.

…..AAAAAAAAAA...

Ma qual’è inoltre il grosso vantaggio che un’azienda dovrebbe considerare nel seguire

un approccio di questo tipo?

Il principale vantaggio da evidenziare nell’adozione di questo modello di sviluppo, sta sicuramente nel fatto che un team di sviluppo è in grado di separare e fornire più agilmente il lavoro ai componenti del gruppo, dato che il framework stesso nasce con l’idea di garantire questa separazione di ambiti.

Garantisce sicuramente uno standard nella modalità di lavoro, questo permette certamente l’inserimento di una nuova risorsa in tempi più brevi.

Snippets e modalità di utilizzo

<creazione api, modello codice backend, … >

***Riferimenti***

[***https://angular.io/***](https://angular.io/)

WebSocket/Socket.io

MongoDB

La scelta di MongoDB, ad oggi uno tra i più conosciuti DBMS non relazionali, nasce dall’esigenza di poter gestire una grossa mole di dati senza uno schema database ben preciso.

Basti pensare ad esempio alla necessità di salvare le statistiche utente che, a fronte di un grosso numero di registrazioni, potrebbero richiedere il salvataggio di un gran numero di statistiche ed informazioni da salvare.

Un futuro sviluppo dell’applicazione potrebbe ad esempio dare la possibilità di visionare un match in corso, fornendo agli spettatori funzionalità di commentare la partita o marchiare con una spunta di gradimento le mosse dei giocatori.

Tutto questo, visto in larga scala, potrebbe sicuramente essere reso più performante denormalizzando le strutture dati in essere e gestendo il tutto senza relazioni tra gli oggetti.

Snippets e modalità di utilizzo

<creazione api, modello codice backend, … >

***Riferimenti***

Ionic

….

Snippets e modalità di utilizzo

<creazione api, modello codice backend, … >

***Riferimenti***

Node.js/Javascript

Nodejs ….

Snippets e modalità di utilizzo

<creazione api, modello codice backend, … >

***Riferimenti***

*https://nodejs.org/it/*

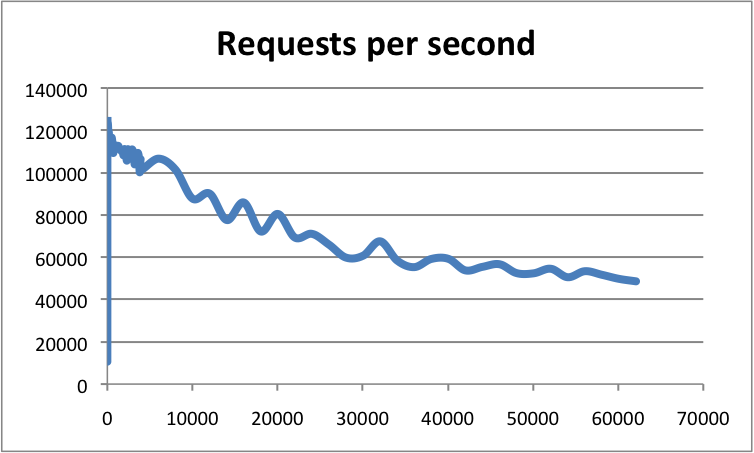
Redis

Redis è fa parte della categoria di prodotti cosiddetti datastore in memory. Sviluppato inizialmente da un italiano, Salvatore Sanfilippo che decide di mantenere il progetto open source (<https://github.com/antirez/redis>), diventa nel breve periodo una community ben più ampia e che oggi probabilmente deve la propria popolarità al gruppo Redis Labs che ha pensato alla commercializzazione di servizi per distribuire il prodotto stesso.

L’adozione di Redis nasce dall’esigenza di persistere in maniera volatile dei dati di gioco in maniera rapida e veloce.

Serve inoltre un meccanismo di publish e subscribe, in grado di notificare/venire a conoscenza delle mosse dei due giocatori per poter poi informare il corrispettivo avversario dell’azione effettuata.

A questo punto, si è scelto di proseguire con la tecnologia in oggetto data la comprovata rapidità nel gestire le richieste, documentata anche dai benchmarks forniti direttamente sul sito del prodotto (<https://redis.io/topics/benchmarks>).



Snippets e modalità di utilizzo

<code snippet>

***Riferimenti***

[*https://github.com/noderedis/node\_redis*](https://github.com/noderedis/node_redis)

*https://redis.io/*

### PaaS – Sistema di distribuzione ed hosting della piattaforma

Arrivati a questo punto, c’è la necessità di capire come distribuire e rendere disponibile in tutta internet il nostro servizio.

Chiaramente, date le tecnologie utilizzate, abbiamo bisogno di un ecosistema architetturale abbastanza complesso, che metta a disposizione tutta una serie di servizi (database, compiler node, memory cache, ecc…) e che soprattutto sia in grado di configurare i vari prodotti in uso senza rendere troppo complesse le interoperabilità tra i servizi scelti.

A questo punto l’unica soluzione al momento disponibile è adottare un servizio PaaS (Product as a Service) che pensi al posto nostro a configurare e rendere operativo l’intero parco applicativo.

PaaS

Pubblicazione del proprio prodotto all'interno di un'applicazione hosting che si preoccuperà di distribuire e rendere fruibile da parte del nostro cliente quanto sviluppato. A questo punto non ci preoccupiamo dell'infrastruttura di rete, delle configurazioni di bilanciamento tra server e di qualsiasi ammenicolo legato a macchine virtuali o altro.

Heroku

Tra le varie opportunità sul mercato, la soluzione proposta da Heroku sembra la più appetibile in termini di costi/benefici.

Si basa su sistemi UNIX.

Struttura:

**Routers**: gestione del routing dei servizi, redirezionano le chiamate al corretto gestore. Praticamente è uno zoccolo scritto in Erlang (Hermes) che fa da bridge tra la nostra applicazione e il dyno. Generalmente gestisce un timeout di 30 secondi classico per il routing della risorsa richiesta.

**Dynos**: potremmo vederla come una macchina virtuale o un virtual server che esegue il nostro codice,

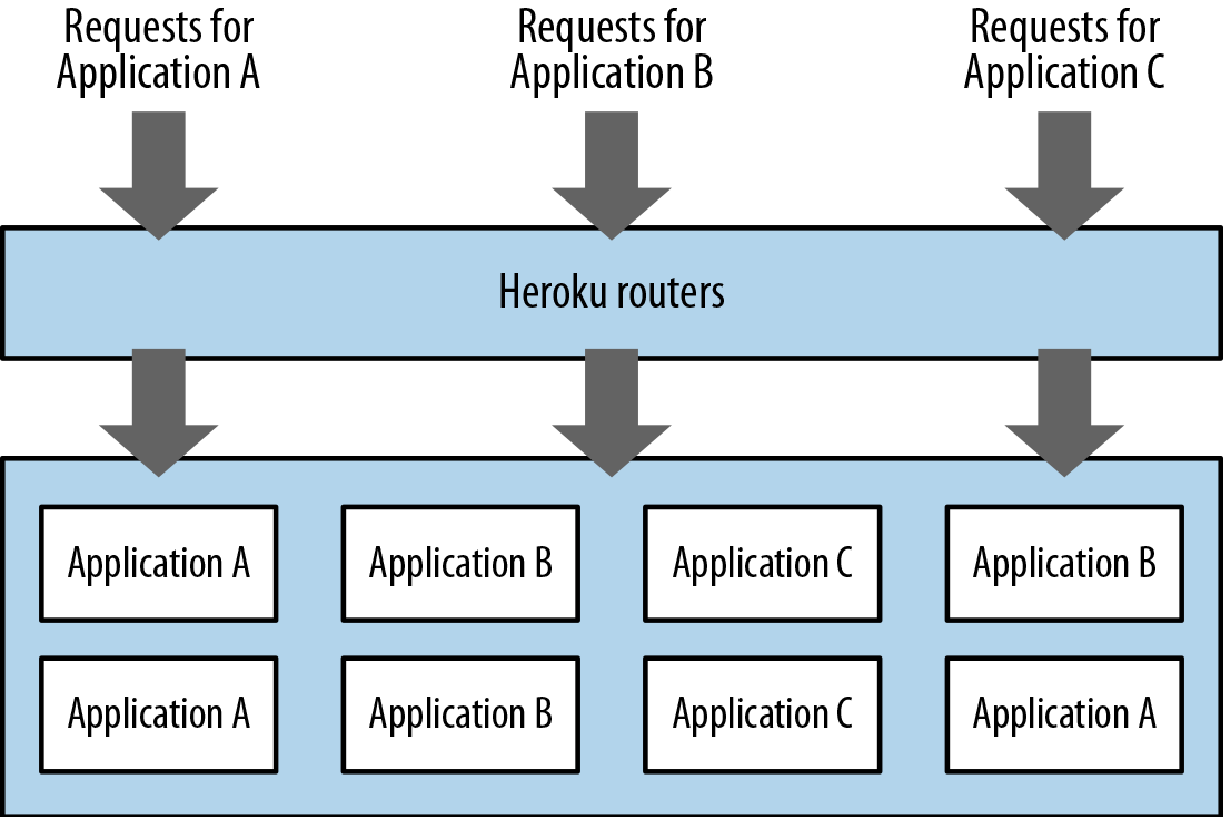
generalmente un dyno viene fornito con 512RAM . Le applicazioni vengono appunto principalmente fatte pagare per dyno/ora con un dettaglio al secondo.

Chiaramente più paghi, più è performante il dyno e più dyno compri, più richieste concorrenti riesci a gestire.

Ogni dyno viene riavviato ogni 24 ore.

**Componenti aggiuntivi**: tutto ciò che viene fornito come componente software (servizio) da pagare a parte (vedi MongoDB, Redis, ecc...).

Il funzionamento generale prevede il commit attraverso git del software, questo una volta ricevuto da Heroku, provvede ad incapsularlo in un codice comprensibile da Heroku stesso in gergo definito come “slug”. Il sistema di routing invece, rende trasparente allo sviluppatore l’astrazione tra il linguaggio e l’application server (Tomcat, Unicorn, ...) che si occupa del dispaccio dei messaggi http verso i client. Il risultato è quello di permettere allo sviluppatore di scrivere il codice con il linguaggio per lui più consono e conosciuto, delegando poi alla piattaforma il compito di distribuirlo e renderlo operativo.



Gestione fault tolerance automatica tra i dyno.

Il grosso vantaggio inoltre offerto da un sistema di questo tipo, è la scalabilità offerta dai “contenitori” (dyno) che ospitano il codice dello sviluppatore; in poche parole, è sufficiente investire più denaro sulla piattaforma per ottenere più potenza di calcolo e memoria da garantire al proprio software. Come se non bastasse, a fronte di un maggior numero di richieste in ingresso, l’acquisto di più contenitori permette di portare il proprio applicativo verso un’architettura a microservizi, che se scritto opportunamente, potrà essere replicato per poter far fronte al bisogno di gestire le crescenti richieste verso il proprio sistema.

***Riferimenti***

*https://www.heroku.com/*

# Interfacce

Descrivere le principali interfacce utente dell'applicazione. È importante evidenziare quale stile sarà applicato alle interfacce e quali risorse saranno coinvolte nella loro esecuzione.

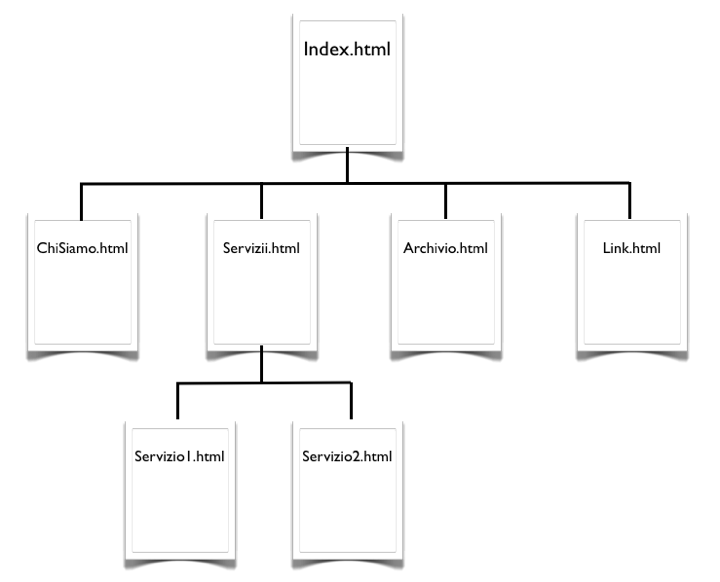
Le interfacce possono essere disegnate o tramite un software o semplicemente a mano.

|  |  |
| --- | --- |
|  | In caso di primo accesso o sessione scaduta, l’utente deve autenticarsi attraverso questa schermata.  La form prevede inoltre un pulsante per la registrazione di un nuovo utente. |
|  |  |
|  |  |
|  |  |
|  |  |

# Architettura

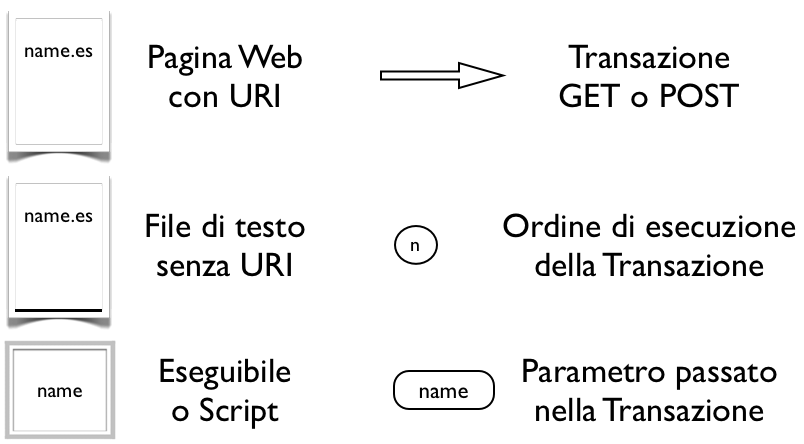
## Diagramma dell'ordine gerarchico delle risorse

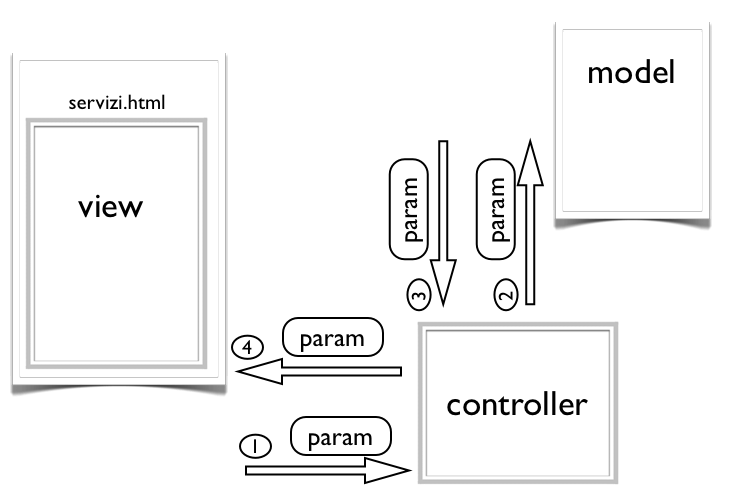
Identificazioni delle pagine da rendere accessibili come risorse (URI) e loro organizzazione gerarchica.

Figura 1: Architettura del sito

## Descrizione delle risorse

Ogni singola risorsa deve essere descritta per le sue componenti MVC indicando le transazioni, l'ordine delle transazioni e gli eventuali parametri passati nella transazione.

Figura 2: Legenda della notazione utilizzata

Figura 3: Componenti MVC della pagina servizi.html

## Altri diagrammi

*Eventuali altri diagrammi, come ad esempio il diagramma del DB*

# Codice

*Frammenti del codice più significativo*

## HTML

## CSS

## API

## Node.js

# Conclusioni

## Heading 1

text

## Heading 2

|  | column 2 | column 3 | column 4 |
| --- | --- | --- | --- |
| item 1 | text text text | text text text text text | R 12 581.36 |
| item 2 | text text text text text | text text text text text | R 5 149.35 |
|  |  |  | R 25.60 |
|  |  | Total due: | \*\* Expression is faulty \*\* |

# Nota bibliografica e sitografica

*Dove si elencano le risorse bibliografiche e sitografiche usate come riferimento e documentazione del lavoro.*

1. Autore, Titolo, Editore, Anno
2. URL, (autore data), Data di consultazione, (URN)